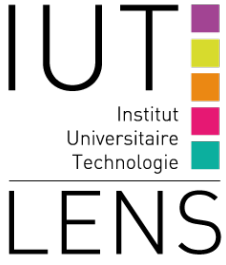
Programmation Web côté serveur

PHP Laravel BD



Fred Hémery

2018/2019

IUT Lens

Département Informatique

Configuration

- La configuration pour un accès à une base de données est faite dans le fichier `config/database.php`.
- Il est possible d'utiliser les SGBDs suivants :
  - MySQL
  - PostgreSQL
  - SQLite
  - SQL Server
- Le fichier `.env` surcharge la configuration du fichier `config/database.php`.
- un exemple de configuration dans le fichier `.env` pour un accès à un SGBD MySQL :

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravelBD
DB_USERNAME=root
DB_PASSWORD=secret
```

IUT Institut Universitaire Technologie LENS

Requêtes brutes

- Laravel propose une façade `DB` qui permet d'exécuter des requêtes SQL, `insert`, `update`, `select`, `delete`.

```php
<?php
namespace App\Http\Controllers;

use Illuminate\Support\Facades\DB;
use App\Http\Controllers\Controller;

class UserController extends Controller {
    public function index() {
        $users = DB::select('select * from users where active = ?', [1]);

        return view('user.index', ['users' => $users]);
    }
}
```

- Le résultat d'une requête `DB::select()` est un tableau. Chaque élément du tableau est une instance de classe `StdClass`.

Requêtes brutes

- Les requêtes brutes de type `insert`, `update`, `delete` renvoient le nombre d'enregistrements impactés.

```php
<?php
namespace App\Http\Controllers;

use Illuminate\Support\Facades\DB;
use App\Http\Controllers\Controller;

class UserController extends Controller {
    public function insert() {
        DB::insert('insert into users (id, name) values (?, ?)', [1, '
        ➥Dayle']);
    }

    public function update() {
      $affected = DB::update('update users set votes = 100 where name = ?'
      ➥, ['John']);
    }

    public function delete() {
      $deleted = DB::delete('delete from users');
    }
}
```

- Les requêtes sans résultat

```php
DB::statement('drop table users');
```

Utilisation du Query Builder

- Laravel propose une classe `Query Builder` qui sécurise les requêtes et qui peut être contrainte afin d'obtenir le résultat.

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Support\Facades\DB;
use App\Http\Controllers\Controller;

class UserController extends Controller {
    public function index() {
        $users = DB::table('users')->get();
        return view('user.index', ['users' => $users]);
    }
   public function show() {
     $user = DB::table('users')->where('name', 'John')->first();
   }
   public function email() {
     $email = DB::table('users')->where('name', 'John')->value('email');
   }
   public function nbUsers() {
     $users = DB::table('users')->count();
   }
   public function nbUsers() {
     $price = DB::table('orders')->where('finalized', 1)->avg('price');
   }
}
```

Utilisation du Query Builder

- Laravel propose une classe `Query Builder` qui sécurise les requêtes et qui peut être contrainte afin d'obtenir le résultat.

```php
<?php

namespace App\Http\Contro

use Illuminate\Support\Facades\DB;
use App\Http\Controllers\Controller;

class UserController extends Controller {
    public function index() {
        $users = DB::table('users')->get();
        return view('user.index', ['users' => $users]);
    }
   public function show() {
     $user = DB::table('users')->where('name', 'John')->first();
   }
   public function email() {
     $email = DB::table('users')->where('name', 'John')->value('email');
   }
   public function nbUsers() {
     $users = DB::table('users')->count();
   }
   public function nbUsers() {
     $price = DB::table('orders')->where('finalized', 1)->avg('price');
   }
}
```

`DB::table('users')` est une instance de la classe `Query`

IUT Institut Universitaire Technologique LENS

Utilisation du Query Builder

- Laravel propose une classe `Query Builder` qui sécurise les requêtes et qui peut être contrainte afin d'obtenir le résultat.

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Support\Facades\DB;
use App\Http\Controllers\Controller;

class UserController extends Controller {
    public function index() {
        $users = DB::table('users')->get();
        return view('user.index', ['users' => $users]);
    }
   public function show() {
                                                    ere('name', 'John')->first();


                                                    here('name', 'John')->value('email');


                                                    ount();
   }
   public function nbUsers() {
     $price = DB::table('orders')->where('finalized', 1)->avg('price');
   }
}
```

`get()` renvoie le résultat de type `Illuminate\Support\Collection`. Chaque élément est une instance de la classe `StdClass`.

Utilisation du Query Builder

- Laravel propose une classe `Query Builder` qui sécurise les requêtes et qui peut être contrainte afin d'obtenir le résultat.

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Support\Facades\DB;
use App\Http\Controllers\Controller;

class UserController extends Controller {
    public function index() {
        $users = DB::table('users')->get();
        return view('user.index', ['users' => $users]);
    }
   public function show() {
     $user = DB::table('users')->where('name', 'John')->first();
   }
   public function email() {
     $email = DB::table('users')->where('name', 'John')->value('email');
   }
   public function nbUsers() {
     $users = DB::table('users')->count();
   }
   public function nbUsers() {
     $price = DB::table('orders')->where('finalized', 1)->avg('price');
   }
}
```

Une contrainte sur la colonne `name`

Utilisation du Query Builder

- Laravel propose une classe `Query Builder` qui sécurise les requêtes et qui peut être contrainte afin d'obtenir le résultat.

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Support\Facades\DB;
use App\Http\Controllers\Controller;

class UserController extends Controller {
    public function index() {
        $users = DB::table('users')->get();
        return view('user.index', ['users' => $users]);
    }
   public function show() {
     $user = DB::table('users')->where('name', 'John')->first();
   }
   public function email() {
     $email = DB::table('users')->where('name', 'John')->value('email');
   }
   public function nbUsers() {
     $users = DB::table('users')->count();
   }
   public function nbUsers() {
     $price = DB::table('orders')->where('finalized', 1)->avg('price');
   }
}
```

Le premier enregistrement

Utilisation du Query Builder

- Laravel propose une classe `Query Builder` qui sécurise les requêtes et qui peut être contrainte afin d'obtenir le résultat.

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Support\Facades\DB;
use App\Http\Controllers\Controller;

class UserController extends Controller {
    public function index() {
        $users = DB::table('users')->get();
        return view('user.index', ['users' => $users]);
    }
   public function show() {
     $user = DB::table('users')->where('name', 'John')->first();
   }
   public function email() {
     $email = DB::table('users')->where('name', 'John')->value('email');
   }
   public function nbUsers() {
     $users = DB::table('users')->count();
   }
   public function nbUsers() {
     $price = DB::table('orders')->wher
   }
}
```

Projection qui ne renvoie que la valeur de la colonne `email`

Utilisation du Query Builder

- Laravel propose une classe `Query Builder` qui sécurise les requêtes et qui peut être contrainte afin d'obtenir le résultat.

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Support\Facades\DB;
use App\Http\Controllers\Controller;

class UserController extends Controller {
    public function index() {
        $users = DB::table('users')->get();
        return view('user.index', ['users' => $users]);
    }
   public function show() {
     $user = DB::table('users')->where('name', 'John')->first();
   }
   public function email() {
     $email = DB::table('users')->where('name', 'John')->value('email');
   }
   public function nbUsers() {
     $users = DB::table('users')->count();
   }
   public function nbUsers() {
     $price = DB::table('orders')->where('finalized', 1)->avg('price');
   }
}
```

Utilisation d'une fonction agrégat

Utilisation du Query Builder

- Laravel propose une classe `Query Builder` qui sécurise les requêtes et qui peut être contrainte afin d'obtenir le résultat.

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Support\Facades\DB;
use App\Http\Controllers\Controller;

class UserController extends Controller {
    public function index() {
        $users = DB::table('users')->get();
        return view('user.index', ['users' => $users]);
    }
   public function show() {
     $user = DB::table('users')->where('name', 'John')->first();
   }
   public function email() {
     $email = DB::table('users')->where('
   }
   public function nbUsers() {
     $users = DB::table('users')->count();
   }
   public function nbUsers() {
     $price = DB::table('orders')->where('finalized', 1)->avg('price');
   }
}
```

Utilisation d'une fonction agrégat après une sélection

Utilisation du Query Builder

- Exemple d'une jointure

```
$users = DB::table('users')
            ->join('contacts', 'users.id', '=', 'contacts.user_id')
            ->join('orders', 'users.id', '=', 'orders.user_id')
            ->select('users.*', 'contacts.phone', 'orders.price')
            ->get();
```

- Exemple de selection

```
$users = DB::table('users')->where([
    ['status', '=', '1'],
    ['subscribed', '<>', '1'],
])->get();
```

- Exemple de tri

```
$users = DB::table('users')
                ->orderBy('name', 'desc')
                ->get();
```

- Exemple de `group by`

```
$users = DB::table('users')
                ->groupBy('account_id')
                ->having('account_id', '>', 100)
                ->get();
```

IUT Institut Universitaire Technologique LENS

Utilisation du Query Builder insert, update delete

- Exemple d'insertion

```
DB::table('users')->insert(
    ['email' => 'johnexample.com', 'votes' => 0]);
```

- Exemple de modification

```
DB::table('users')
            ->where('id', 1)
            ->update(['votes' => 1]);
```

- Exemple de suppression

```
DB::table('users')->where('votes', '>', 100)->delete();
```

Introduction

- Laravel propose un mécanisme qui faire correspondre à chaque table d'une base de données un modèle (classe PHP).
- Un modèle est une classe qui hérite de la classe

`Illuminate\Database\Eloquent\Model`

Example

```php
<?php
namespace App;

use Illuminate\Database\Eloquent\Model;
class Flight extends Model{

    protected $table = 'my_flights'; // par défaut
    ➥flights
    protected $primaryKey = 'id_flight'; // par défaut id
    public $timestamps = false; // par défaut true
}
```

Un modèle comme interface du résultat d'une requête

Example

```php
<?php

use App\Flight;
$flights = App\Flight::all();

foreach ($flights as $flight) {
    echo $flight->name;
}
$flights = App\Flight::where('active', 1)
               ->orderBy('name', 'desc')
               ->take(10)
               ->get();
```

- Les résultats sont donnés sous forme d'une collection :

  `Illuminate\Database\Eloquent\Collection`

  qui dispose de nombreuses méthodes.

Résultat unique et agrégat

- La fonction statique `all()` de la classe modèle et la fonction `get()` retourne une collection
- Les fonctions `first()` et `find()` retourne un résultat (ou un tableau de résultats pour `find()`)
- Les fonctions agrégats `count`, `sum`, `max`, ...

Example

```
// premier enr. qui correspond
$flight = App\Flight::where('active', 1)->first();

// renvoie l'enr. avec la clé 1
$flight = App\Flight::find(1);
// renvoie un tableau contenant les enr. qui ont les clés 1,2,3
$flights = App\Flight::find([1, 2, 3]);
// compte le nombre d'enr.
$count = App\Flight::all()->count();
// renvoie la valeur max
$max = App\Flight::where('active', 1)->max('price');
```

IUT Institut Universitaire Technologique LENS

Création, modification d'enregistrements

- L'insertion d'un enregistrement se fait en créant une instance de la classe du modèle puis en appelant la méthode `save()`.

Création

```
$flight = new Flight;
$flight->name = $request->name;
$flight->save();
```

- La modification d'un enregistrement se fait en récupérant un enregistrement ou en utilisant la fonction `update()`

Modification

```
$flight = Flight::find(1);
$flight->name = 'New Flight Name';
$flight->save();
```

Modification

```
Flight::where('active', 1)
  ->where('destination', 'Ajaccio')
  ->update(['delayed' => 1]);
```

IUT LENS

Suppression d'enregistrements

- La suppression d'un enregistrement se fait en récupérant un enregistrement ou en utilisant la fonction `destroy()`

Suppression

```
$flight = Flight::find(1);

$flight->delete();
```

Suppression

```
Flight::destroy(1);

Flight::destroy([1, 2, 3]);

Flight::destroy(1, 2, 3);
```

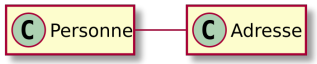
- On peut supprimer des enregistrements résultat d'une requête

Suppression

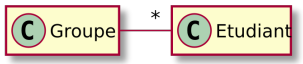
```
$deletedRows = App\Flight::where('active', 0)->delete();
```

Les différentes associations entre entités

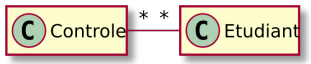
One to One une instance d'entité est en relation avec une et une seule instance d'entité (une personne à une adresse)



One to Many une instance d'entité est en relation avec plusieurs instances de l'autre entité (un groupe est composé de plusieurs étudiants)



Many to Many plusieurs instances d'un entité sont en relation avec plusieurs instances de l'autre entité (un étudiants rédige plusieurs contrôle et un contrôle est composé par plusieurs étudiants)



- Chacune des associations précédentes peuvent être
  - uni-directionnelle : seule l'une des entités connait l'association
  - bi-directionnelle : les deux entités, en association, ont connaissance de l'association

IUT Institut Universitaire Technologie LENS

Association One to One

exemple repris de

`https://laravel.com/docs/5.5/eloquent-relationships#one-to-one`

- la fonction `hasOne()` permet d'indiquer l'entité à l'autre extrémité de l'association

```
<?php
namespace App;
use Illuminate\Database\Eloquent\Model;
class User extends Model {
    public function phone() { return $this->hasOne('App\Phone'); }
}
```

- la fonction considère que dans la table en vis à vis, il y a une clé étrangère qui pointe vers la clé primaire de la table courante.
- Le nom de la clé étrangère doit être composé du nom de la table (dans l'exemple `user`) suivi du nom de la clé primaire local (`_id`) pour former le nom complet (dans l'exemple `user_id`).
- Il est possible de modifier les noms par défaut en ajoutant des paramètres à la fonction `hasOne()`.

```
public function phone() { return $this->hasOne('App\Phone', 'foreign_key', '
    ➥local_key'); }
```

IUT Institut Universitaire Technologique LENS

Association One to One inverse

- La relation inverse : la fonction `belongsTo()` permet d'indiquer l'entité à l'autre extrémité de l'association

```php
<?php

namespace App;
use Illuminate\Database\Eloquent\Model;
class Phone extends Model{
    public function user() {
        return $this->belongsTo('App\User');
    }
}
```

- Cela signifie que la table dispose d'une colonne (`user_id` dans l'exemple) qui est une clé étrangère qui pointe vers la clé primaire (`id`) de la table en association (`user`)
- Il est possible de surcharger ces valeurs par défaut

```php
        return $this->belongsTo('App\User', 'foreign_key', 'other_key');
}
```

IUT Institut Universitaire Technologique LENS

Association One to Many

exemple repris de

`https://laravel.com/docs/5.5/eloquent-relationships#one-to-many`

- la fonction `hasMany()` permet d'indiquer l'entité à l'autre extrémité de l'association

```php
<?php
namespace App;
use Illuminate\Database\Eloquent\Model;
class Post extends Model {
    public function comments() { return $this->hasMany('App\Comment'); }
}
```

- la fonction considère que dans la table en vis à vis, il y a une clé étrangère qui pointe vers la clé primaire de la table courante.
- Comme pour l'association One To One, Laravel utilise des valeurs par défaut pour retrouver la clé étrangère. Ces valeurs peuvent aussi être surchargées.
- On peut faire une sélection dans la collection en résultat

```php
$comments = App\Post::find(1)->comments;
foreach ($comments as $comment) {
    //
}

$comments = App\Post::find(1)->comments()->where('title', 'foo')->first();
```

IUT Institut Universitaire Technologie LENS

Association One to Many inverse

- la relation inverse est obtenu par la fonction `belongsTo()` permet d'indiquer l'entité à l'autre extrémité de l'association
- comme dans l'association précédente, les valeurs par défaut peuvent être surchargées.

```php
<?php
namespace App;

use Illuminate\Database\Eloquent\Model;

class Comment extends Model {
  public function post() {
        return $this->belongsTo('App\Post');
  }
}
```

- Après déclaration de la fonction, il est possible d'utiliser l'association dans l'autre sens

```php
$comment = App\Comment::find(1);

echo $comment->post->title;
```

IUT Institut Universitaire Technologie LENS

Association Many To Many

exemple repris de

`https://laravel.com/docs/5.5/eloquent-relationships#many-to-many`

- Dans cette association, une table intermédiaire est utilisée pour sauvegarder les enregistrements en relation.
- la fonction `belongsToMany()` permet d'indiquer le nom de l'entité à l'autre extrémité de l'association

```php
<?php

namespace App;
use Illuminate\Database\Eloquent\Model;
class User extends Model {
    public function roles() {
        return $this->belongsToMany('App\Role');
    }
}
```

- Le nom de la table intermédiaire est la concaténation des deux tables (triées par ordre lexicographique) en relation (`role_user`), cette valeur peut être surchargée.

IUT Institut Universitaire Technologie LENS

Association Many To Many inverse

- L'inverse de l'association Many To Many , utilise la même fonction `belongsToMany()`.

```php
<?php
namespace App;
use Illuminate\Database\Eloquent\Model;
class Role extends Model {
  public function users()  {
        return $this->belongsToMany('App\User');
  }
}
```

La table intermédiaire de l'association Many To Many

- Pour avoir un accès à la table intermédiaire, on utilise la fonction `pivot()`.

```php
$user = App\User::find(1);

foreach ($user->roles as $role) {
    echo $role->pivot->role_id;
}
```

- Par défaut la table intermédiaire ne contient que les clés des entités en association.
- On peut ajouter les champs `created_at` et `modified_at` avec la fonction `withTimestamps()`.

```php
return $this->belongsToMany('App\Role')->withTimestamps();
```

- On peut ajouter des champs supplémentaires.

```php
return $this->belongsToMany('App\Role')->withPivot('column1', 'column2');
```

- On peut filtrer en utilisant une colonne de la table intermédiaire

```php
return $this->belongsToMany('App\Role')->wherePivot('approved', 1);
return $this->belongsToMany('App\Role')->wherePivotIn('priority', [1, 2]);
```

IUT Institut Universitaire Technologie LENS

Interrogations à l'aide des associations

- Existence d'une ou ou plusieurs entités associées

```php
$posts = App\Post::has('comments')->get();
$posts = Post::has('comments', '>=', 3)->get();
$posts = Post::has('comments.votes')->get();
// Renvoie les post(s) avec au moins un comment qui contient le mot qui
    ➥commence par foo
$posts = Post::whereHas('comments', function ($query) {
    $query->where('content', 'like', 'foo%');
})->get();
```

- Absence d'une ou ou plusieurs propriétés de l'entité associée

```php
$posts = App\Post::doesntHave('comments')->get();

$posts = Post::whereDoesntHave('comments', function ($query) {
    $query->where('content', 'like', 'foo%');
})->get();
```

- La fonction `withCount()` permet de récupérer le nombre d'entités en association sans les lire.

```php
$posts = App\Post::withCount('comments')->get();

foreach ($posts as $post) {
    echo $post->comments_count;
}
```

Chargement différé

- Par défaut les entités en relation sont récupérées uniquement lorsqu'on les utilise. Dans l'exemple suivant (tiré de `https://laravel.com/docs/5.5/eloquent-relationships`)

```
namespace App;
use Illuminate\Database\Eloquent\Model;
class Book extends Model {
    public function author()  {
        return $this->belongsTo('App\Author');
    }
}
```

- Lecture avec chargement différé.

```
$books = App\Book::all();
foreach ($books as $book) {
    echo $book->author->name;
}
```

- Lecture sans chargement différé et utilisation de la fonction `with()`.

```
$books = App\Book::with('author')->get();
foreach ($books as $book) {
    echo $book->author->name;
}
```

IUT Institut Universitaire Technologique LENS

Sauvegarde des entités en association

- Sauvegarde d'une entité en conséquence de son association avec une autre entité.

```
$comment = new App\Comment(['message' => 'A new comment.']);
$post = App\Post::find(1);
$post->comments()->save($comment);
```

- La fonction `saveMany([...])` sera utilisée pour sauvegarder un tableau d'entités.

```
$post = App\Post::find(1);

$post->comments()->saveMany([
    new App\Comment(['message' => 'A new comment.']),
    new App\Comment(['message' => 'Another comment.']),
]);
```

Modification de l'association entre entités

- Modification de l'association à partir de l'entité fille à l'aide de la fonction `associate()` (modification de la clé étrangère).

```
$account = App\Account::find(10);

$user->account()->associate($account);

$user->save();
```

La classe `User` (entité fille) s'associe avec la classe mère `Account`.

- Suppression de l'association à partir de l'entité fille à l'aide de la fonction `dissociate()` (suppression de la clé étrangère).

```
$user->account()->dissociate();

$user->save();
```

Modification de l'association entre entités : cas Many To Many

- Dans le cas de l'association entre un utilisateur (`User`) et ses rôles (`Role`) exemple tiré de `https://laravel.com/docs/5.5/eloquent-relationships`.
- Pour ajouter un rôle à l'utilisateur on utilisera la méthode `attach()`.

```
$user = App\User::find(1);

$user->roles()->attach($roleId);
```

- Pour supprimer un rôle on utilisera la fonction `detach()`.

```
// Detach a single role from the user...
$user->roles()->detach($roleId);

// Detach all roles from the user...
$user->roles()->detach();
```